FOUNDATION-0.2e: Application Schema - PART 1 (Code)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 2 Afternoon)

Component: Application Agent Database Schema

Estimated Time: 15 min AI execution + 15 min verification

Complexity: MEDIUM

Risk Level: LOW

Files: Part 1 of 2 (Code implementation)

MILESTONE: This completes the PostgreSQL schema (100%)! 🎉

📦 DEPENDENCIES

Must Complete First:

- **FOUNDATION-0.2a:** Core Database Schema ✅ COMPLETED
- **FOUNDATION-0.2b:** Agent State Tables ✅ COMPLETED
- **FOUNDATION-0.2c:** Workflow History Tables ✅ COMPLETED
- **FOUNDATION-0.2d:** Resource Schema Tables ✅ COMPLETED

Required Services Running:

```bash
# Verify all services are healthy
cd ~/optiinfra
make verify

# Expected output:
# PostgreSQL... ✅ HEALTHY
# ClickHouse... ✅ HEALTHY
# Qdrant... ✅ HEALTHY
# Redis... ✅ HEALTHY
```

Database State:

```bash
```

```
# Verify existing tables
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt"

# Expected: 15 tables from 0.2a + 0.2b + 0.2c + 0.2d
```

🎯 OBJECTIVE

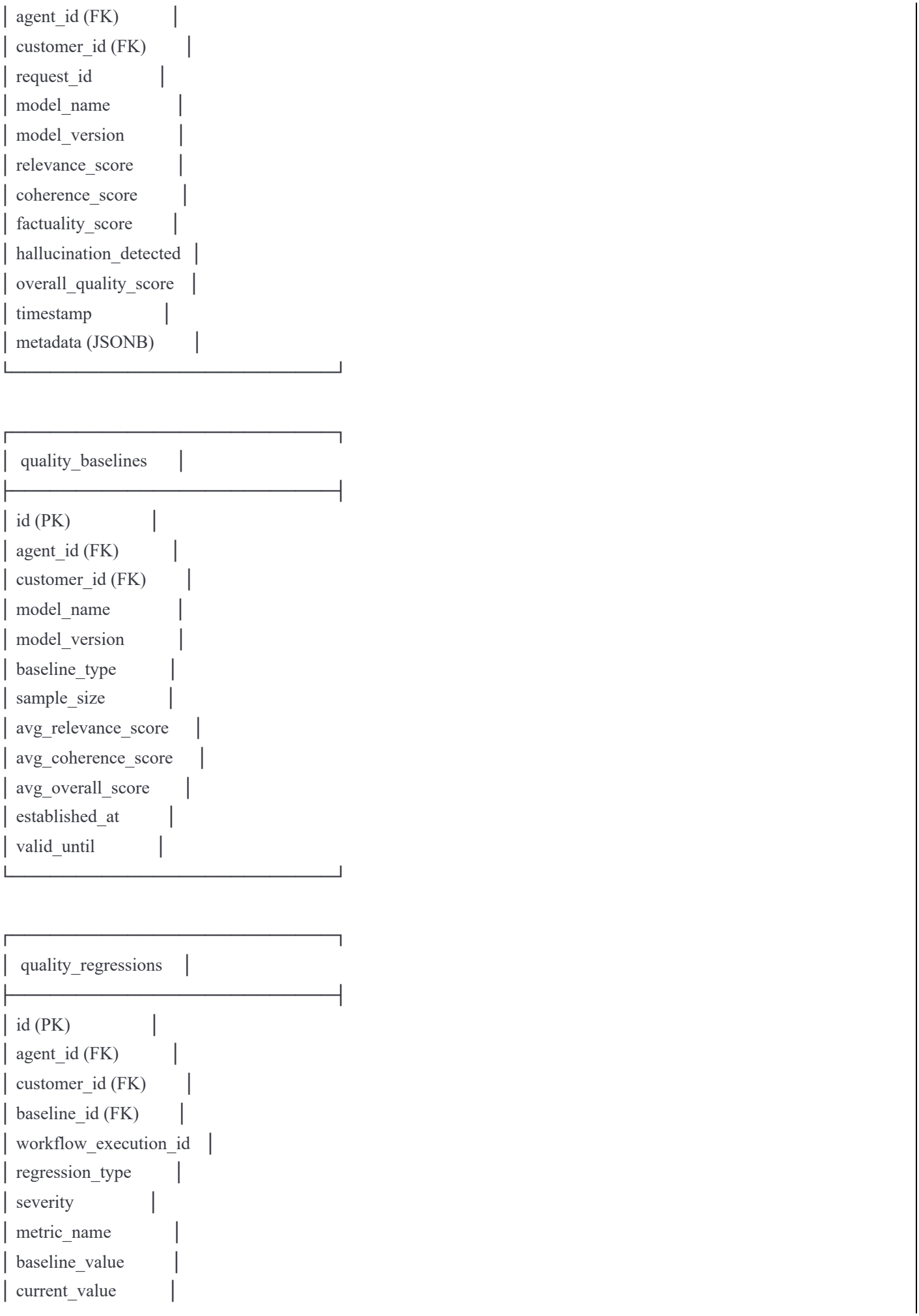
Add **Application Agent-specific database tables** to track LLM output quality, baselines, and regressions. These tables enable:

1. **Quality Metrics** - Track per-request quality scores (relevance, coherence, factuality, hallucinations)
2. **Quality Baselines** - Establish and maintain quality baselines per model
3. **Quality Regressions** - Detect and track quality degradations

What This Enables:

- ✅ Monitor LLM output quality in real-time
- ✅ Detect hallucinations and toxicity
- ✅ Establish quality baselines per model/version
- ✅ Automatically detect quality regressions
- ✅ Trigger rollbacks when quality drops
- ✅ Prevent bad optimizations from reaching production
- ✅ A/B testing with quality validation

Database Design:


agent_id (FK)
customer_id (FK)
request_id
model_name
model_version
relevance_score
coherence_score
factuality_score
hallucination_detected
overall_quality_score
timestamp
metadata (JSONB)
quality_baselines
id (PK)
agent_id (FK)
customer_id (FK)
model_name
model_version
baseline_type
sample_size
avg_relevance_score
avg_coherence_score
avg_overall_score
established_at
valid_until
quality_regressions
id (PK)
agent_id (FK)
customer_id (FK)
baseline_id (FK)
workflow_execution_id
regression_type
severity
metric_name
baseline_value
current_value

```
| delta_percent          |
| action_taken           |
| resolved_at            |
└────────────────────────────────────────┘
```

📁 FILE 1: Application Schema Models

Location: `~/optiinfra/shared/database/models/application_schema.py`

```python
```

```
"""
Application Schema Models (FOUNDATION-0.2e)
Tracks LLM quality metrics, baselines, and regressions
"""

from datetime import datetime
from sqlalchemy import (
    Column, String, DateTime, Float, Integer, Boolean, Text,
    Enum as SQLEnum, ForeignKey, Index
)
from sqlalchemy.dialects.postgresql import UUID, JSONB
from sqlalchemy.orm import relationship
import uuid
import enum

from shared.database.base import Base


# ============================================
# ENUMS
# ============================================

class BaselineType(str, enum.Enum):
    """Types of quality baselines"""
    INITIAL = "initial"
    UPDATED = "updated"
    ROLLBACK = "rollback"


class RegressionType(str, enum.Enum):
    """Types of quality regressions"""
    QUALITY_DROP = "quality_drop"
    LATENCY_INCREASE = "latency_increase"
    HALLUCINATION_SPIKE = "hallucination_spike"
    TOXICITY_INCREASE = "toxicity_increase"


class RegressionSeverity(str, enum.Enum):
    """Severity levels for regressions"""
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"
    CRITICAL = "critical"
```

```
class RegressionAction(str, enum.Enum):
    """Actions taken for regressions"""
    ALERT_ONLY = "alert_only"
    ROLLBACK_TRIGGERED = "rollback_triggered"
    MANUAL_REVIEW = "manual_review"
    AUTO_FIXED = "auto_fixed"


# ============================================
# MODELS
# ============================================

class QualityMetric(Base):
    """
    Tracks per-request quality metrics for LLM outputs

    Stores detailed quality scores for each LLM request to enable
    quality monitoring, regression detection, and A/B testing.
    """
    __tablename__ = "quality_metrics"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Keys
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    customer_id = Column(
        UUID(as_uuid=True),
        ForeignKey("customers.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
```

```
    # Request Details
    request_id = Column(
        String(255),
        nullable=False,
        unique=True,
        index=True,
        comment="Unique identifier for the LLM request"
    )
    model_name = Column(
        String(100),
        nullable=False,
        index=True,
        comment="Model name: gpt-4, claude-3-opus, llama-3-70b, etc."
    )
    model_version = Column(
        String(50),
        nullable=False,
        comment="Model version: 0314, 20240229, etc."
    )

    # Token Usage
    prompt_tokens = Column(
        Integer,
        nullable=True,
        comment="Number of tokens in prompt"
    )
    completion_tokens = Column(
        Integer,
        nullable=True,
        comment="Number of tokens in completion"
    )

    # Performance
    latency_ms = Column(
        Float,
        nullable=True,
        comment="Request latency in milliseconds"
    )

    # Quality Scores (0.0 to 1.0)
    relevance_score = Column(
        Float,
        nullable=True,
```

```
    comment="How relevant is the response to the prompt (0-1)"
)
coherence_score = Column(
    Float,
    nullable=True,
    comment="How coherent and logical is the response (0-1)"
)
factuality_score = Column(
    Float,
    nullable=True,
    comment="How factually accurate is the response (0-1)"
)
hallucination_detected = Column(
    Boolean,
    nullable=False,
    default=False,
    index=True,
    comment="Whether hallucination was detected"
)
toxicity_score = Column(
    Float,
    nullable=True,
    comment="Toxicity level of the response (0-1, lower is better)"
)
overall_quality_score = Column(
    Float,
    nullable=False,
    index=True,
    comment="Aggregated quality score (0-1)"
)

# Timing
timestamp = Column(
    DateTime,
    nullable=False,
    index=True,
    comment="When the request was made"
)

# Additional Data
metadata = Column(
    JSONB,
    nullable=False,
    default=dict,
```

```
        comment="Additional context: user_id, session_id, etc."
    )

    # Audit
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    # Relationships
    agent = relationship("Agent", back_populates="quality_metrics")
    customer = relationship("Customer", back_populates="quality_metrics")

    # Indexes
    __table_args__ = (
        Index(
            "ix_quality_metrics_agent_model",
            "agent_id", "model_name"
        ),
        Index(
            "ix_quality_metrics_customer_model",
            "customer_id", "model_name"
        ),
        Index(
            "ix_quality_metrics_timestamp",
            "timestamp"
        ),
        Index(
            "ix_quality_metrics_overall_score",
            "overall_quality_score"
        ),
        Index(
            "ix_quality_metrics_model_timestamp",
            "model_name", "timestamp"
        ),
    )

    def __repr__(self):
        return (
            f"<QualityMetric(id={self.id}, "
            f"model={self.model_name}, score={self.overall_quality_score:.2f})>"
        )


class QualityBaseline(Base):
    """
    Stores quality baselines for comparison and regression detection
```

```
    Baselines are established from sample sets and used to detect
    when quality degrades after changes.
    """
    __tablename__ = "quality_baselines"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Keys
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    customer_id = Column(
        UUID(as_uuid=True),
        ForeignKey("customers.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )

    # Model Details
    model_name = Column(
        String(100),
        nullable=False,
        index=True
    )
    model_version = Column(
        String(50),
        nullable=False
    )

    # Baseline Type
    baseline_type = Column(
        SQLEnum(BaselineType, name="baseline_type"),
        nullable=False,
        default=BaselineType.INITIAL,
```

```
    index=True
)

# Sample Info
sample_size = Column(
    Integer,
    nullable=False,
    comment="Number of requests used to establish baseline"
)

# Average Scores
avg_relevance_score = Column(
    Float,
    nullable=True,
    comment="Average relevance score from samples"
)
avg_coherence_score = Column(
    Float,
    nullable=True,
    comment="Average coherence score from samples"
)
avg_factuality_score = Column(
    Float,
    nullable=True,
    comment="Average factuality score from samples"
)
avg_overall_score = Column(
    Float,
    nullable=False,
    comment="Average overall quality score"
)

# Performance Baseline
p95_latency_ms = Column(
    Float,
    nullable=True,
    comment="P95 latency from samples"
)

# Validity
established_at = Column(
    DateTime,
    nullable=False,
    index=True,
```

```
    comment="When the baseline was established"
)
valid_until = Column(
    DateTime,
    nullable=True,
    comment="When the baseline expires (null = indefinite)"
)

# Additional Data
metadata = Column(
    JSONB,
    nullable=False,
    default=dict,
    comment="Additional context: confidence_interval, std_dev, etc."
)

# Audit
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

# Relationships
agent = relationship("Agent", back_populates="quality_baselines")
customer = relationship("Customer", back_populates="quality_baselines")
regressions = relationship(
    "QualityRegression",
    back_populates="baseline",
    cascade="all, delete-orphan"
)

# Indexes
__table_args__ = (
    Index(
        "ix_quality_baselines_agent_model",
        "agent_id", "model_name"
    ),
    Index(
        "ix_quality_baselines_customer_model",
        "customer_id", "model_name"
    ),
    Index(
        "ix_quality_baselines_established_at",
        "established_at"
    ),
    Index(
        "ix_quality_baselines_type",
```

```
            "baseline_type"
        ),
    )

    def __repr__(self):
        return (
            f"<QualityBaseline(id={self.id}, "
            f"model={self.model_name}, type={self.baseline_type}, "
            f"score={self.avg_overall_score:.2f})>"
        )


class QualityRegression(Base):
    """
    Tracks detected quality regressions

    Records when quality drops below acceptable thresholds,
    enabling automated rollbacks and alerts.
    """
    __tablename__ = "quality_regressions"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Keys
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    customer_id = Column(
        UUID(as_uuid=True),
        ForeignKey("customers.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    baseline_id = Column(
        UUID(as_uuid=True),
```

```
    ForeignKey("quality_baselines.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Baseline used for comparison"
)
workflow_execution_id = Column(
    UUID(as_uuid=True),
    ForeignKey("workflow_executions.id", ondelete="SET NULL"),
    nullable=True,
    index=True,
    comment="Workflow that caused the regression (if known)"
)

# Regression Details
regression_type = Column(
    SQLEnum(RegressionType, name="regression_type"),
    nullable=False,
    index=True
)
severity = Column(
    SQLEnum(RegressionSeverity, name="regression_severity"),
    nullable=False,
    index=True
)

# Detection Time
detected_at = Column(
    DateTime,
    nullable=False,
    index=True
)

# Metric Details
metric_name = Column(
    String(100),
    nullable=False,
    comment="Which metric regressed: overall_score, relevance, latency, etc."
)
baseline_value = Column(
    Float,
    nullable=False,
    comment="Expected value from baseline"
)
current_value = Column(
```

```
    Float,
    nullable=False,
    comment="Actual value that triggered regression"
)
delta_percent = Column(
    Float,
    nullable=False,
    comment="Percentage change from baseline (negative = worse)"
)
sample_size = Column(
    Integer,
    nullable=False,
    comment="Number of samples used to detect regression"
)

# Action & Resolution
action_taken = Column(
    SQLEnum(RegressionAction, name="regression_action"),
    nullable=False,
    default=RegressionAction.ALERT_ONLY,
    index=True
)
resolved_at = Column(
    DateTime,
    nullable=True,
    comment="When the regression was resolved"
)
resolution_notes = Column(
    Text,
    nullable=True,
    comment="How the regression was resolved"
)

# Additional Data
metadata = Column(
    JSONB,
    nullable=False,
    default=dict,
    comment="Additional context: affected_requests, p_value, etc."
)

# Audit
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
updated_at = Column(
```

```
    DateTime,
    default=datetime.utcnow,
    onupdate=datetime.utcnow,
    nullable=False
)

# Relationships
agent = relationship("Agent", back_populates="quality_regressions")
customer = relationship("Customer", back_populates="quality_regressions")
baseline = relationship("QualityBaseline", back_populates="regressions")
workflow = relationship(
    "WorkflowExecution",
    foreign_keys=[workflow_execution_id],
    backref="quality_regressions"
)

# Indexes
__table_args__ = (
    Index(
        "ix_quality_regressions_agent_severity",
        "agent_id", "severity"
    ),
    Index(
        "ix_quality_regressions_customer_type",
        "customer_id", "regression_type"
    ),
    Index(
        "ix_quality_regressions_detected_at",
        "detected_at"
    ),
    Index(
        "ix_quality_regressions_baseline",
        "baseline_id"
    ),
    Index(
        "ix_quality_regressions_workflow",
        "workflow_execution_id"
    ),
    Index(
        "ix_quality_regressions_unresolved",
        "resolved_at"
    ),
)
```

```
    def __repr__(self):
        return (
            f"<QualityRegression(id={self.id}, "
            f"type={self.regression_type}, severity={self.severity}, "
            f"resolved={self.resolved_at is not None})>"
        )

    @property
    def is_resolved(self):
        """Check if regression has been resolved"""
        return self.resolved_at is not None

    @property
    def time_to_resolve_minutes(self):
        """Calculate time to resolution in minutes"""
        if self.resolved_at and self.detected_at:
            return (self.resolved_at - self.detected_at).total_seconds() / 60
        return None
```

📁 FILE 2: Update Core Models (Add Relationships)

Location: `~/optiinfra/shared/database/models/core.py`

Add these relationships to existing models:

```
python
```

```python
# In the Agent model, add:
from sqlalchemy.orm import relationship

class Agent(Base):
    # ... existing fields ...

    # ADD THESE NEW RELATIONSHIPS:
    quality_metrics = relationship(
        "QualityMetric",
        back_populates="agent",
        cascade="all, delete-orphan"
    )
    quality_baselines = relationship(
        "QualityBaseline",
        back_populates="agent",
        cascade="all, delete-orphan"
    )
    quality_regressions = relationship(
        "QualityRegression",
        back_populates="agent",
        cascade="all, delete-orphan"
    )


# In the Customer model, add:
class Customer(Base):
    # ... existing fields ...

    # ADD THESE NEW RELATIONSHIPS:
    quality_metrics = relationship(
        "QualityMetric",
        back_populates="customer",
        cascade="all, delete-orphan"
    )
    quality_baselines = relationship(
        "QualityBaseline",
        back_populates="customer",
        cascade="all, delete-orphan"
    )
    quality_regressions = relationship(
        "QualityRegression",
        back_populates="customer",
```

```
        cascade="all, delete-orphan"
    )
```

📁 FILE 3: Update Model Imports

Location: `~/optiinfra/shared/database/models/__init__.py`

```python
```

```
"""
Database models package
"""

# Core models (0.2a)
from shared.database.models.core import (
    Customer,
    Agent,
    Event,
    Recommendation,
    Approval,
    Optimization,
    # Enums
    CustomerStatus,
    AgentType,
    AgentStatus,
    EventType,
    EventSeverity,
    RecommendationType,
    ApprovalStatus,
    OptimizationStatus,
)

# Agent state models (0.2b)
from shared.database.models.agent_state import (
    AgentConfig,
    AgentState,
    AgentCapability,
    AgentMetric,
    # Enums
    ConfigType,
    AgentStatusDetail,
    MetricType,
)

# Workflow history models (0.2c)
from shared.database.models.workflow_history import (
    WorkflowExecution,
    WorkflowStep,
    WorkflowArtifact,
    # Enums
    WorkflowType,
    WorkflowStatus,
```

```
    StepStatus,
    ArtifactType,
)

# Resource schema models (0.2d)
from shared.database.models.resource_schema import (
    ResourceMetric,
    ScalingEvent,
    # Enums
    ResourceType,
    ScalingEventType,
)

# Application schema models (0.2e) - NEW!
from shared.database.models.application_schema import (
    QualityMetric,
    QualityBaseline,
    QualityRegression,
    # Enums
    BaselineType,
    RegressionType,
    RegressionSeverity,
    RegressionAction,
)

__all__ = [
    # Core models
    "Customer",
    "Agent",
    "Event",
    "Recommendation",
    "Approval",
    "Optimization",
    # Core enums
    "CustomerStatus",
    "AgentType",
    "AgentStatus",
    "EventType",
    "EventSeverity",
    "RecommendationType",
    "ApprovalStatus",
    "OptimizationStatus",
    # Agent state models
    "AgentConfig",
```

```
    "AgentState",
    "AgentCapability",
    "AgentMetric",
    # Agent state enums
    "ConfigType",
    "AgentStatusDetail",
    "MetricType",
    # Workflow history models
    "WorkflowExecution",
    "WorkflowStep",
    "WorkflowArtifact",
    # Workflow history enums
    "WorkflowType",
    "WorkflowStatus",
    "StepStatus",
    "ArtifactType",
    # Resource schema models
    "ResourceMetric",
    "ScalingEvent",
    # Resource schema enums
    "ResourceType",
    "ScalingEventType",
    # Application schema models - NEW!
    "QualityMetric",
    "QualityBaseline",
    "QualityRegression",
    # Application schema enums - NEW!
    "BaselineType",
    "RegressionType",
    "RegressionSeverity",
    "RegressionAction",
]
```

📁 FILE 4: Alembic Migration

Location: `~/optiinfra/shared/database/migrations/versions/005_application_schema.py`

```
python
```

```
"""Create application schema tables

Revision ID: 005_application_schema
Revises: 004_resource_schema
Create Date: 2025-10-20 18:00:00.000000

"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision = '005_application_schema'
down_revision = '004_resource_schema'
branch_labels = None
depends_on = None


def upgrade() -> None:
    """Create application schema tables"""

    # Create ENUM types
    baseline_type_enum = postgresql.ENUM(
        'initial',
        'updated',
        'rollback',
        name='baseline_type',
        create_type=True
    )
    baseline_type_enum.create(op.get_bind(), checkfirst=True)

    regression_type_enum = postgresql.ENUM(
        'quality_drop',
        'latency_increase',
        'hallucination_spike',
        'toxicity_increase',
        name='regression_type',
        create_type=True
    )
    regression_type_enum.create(op.get_bind(), checkfirst=True)

    regression_severity_enum = postgresql.ENUM(
        'low',
```

```
    'medium',
    'high',
    'critical',
    name='regression_severity',
    create_type=True
)
regression_severity_enum.create(op.get_bind(), checkfirst=True)

regression_action_enum = postgresql.ENUM(
    'alert_only',
    'rollback_triggered',
    'manual_review',
    'auto_fixed',
    name='regression_action',
    create_type=True
)
regression_action_enum.create(op.get_bind(), checkfirst=True)

# Create quality_metrics table
op.create_table(
    'quality_metrics',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('request_id', sa.String(255), nullable=False, unique=True),
    sa.Column('model_name', sa.String(100), nullable=False),
    sa.Column('model_version', sa.String(50), nullable=False),
    sa.Column('prompt_tokens', sa.Integer(), nullable=True),
    sa.Column('completion_tokens', sa.Integer(), nullable=True),
    sa.Column('latency_ms', sa.Float(), nullable=True),
    sa.Column('relevance_score', sa.Float(), nullable=True),
    sa.Column('coherence_score', sa.Float(), nullable=True),
    sa.Column('factuality_score', sa.Float(), nullable=True),
    sa.Column('hallucination_detected', sa.Boolean(), nullable=False, server_default='false'),
    sa.Column('toxicity_score', sa.Float(), nullable=True),
    sa.Column('overall_quality_score', sa.Float(), nullable=False),
    sa.Column('timestamp', sa.DateTime(), nullable=False),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], ondelete='CASCADE'),
)

# Create indexes for quality_baselines
```

```
op.create_index('ix_quality_baselines_id', 'quality_baselines', ['id'])
op.create_index('ix_quality_baselines_agent_id', 'quality_baselines', ['agent_id'])
op.create_index('ix_quality_baselines_customer_id', 'quality_baselines', ['customer_id'])
op.create_index('ix_quality_baselines_model_name', 'quality_baselines', ['model_name'])
op.create_index('ix_quality_baselines_baseline_type', 'quality_baselines', ['baseline_type'])
op.create_index('ix_quality_baselines_established_at', 'quality_baselines', ['established_at'])
op.create_index('ix_quality_baselines_agent_model', 'quality_baselines', ['agent_id', 'model_name'])
op.create_index('ix_quality_baselines_customer_model', 'quality_baselines', ['customer_id', 'model_name'])
op.create_index('ix_quality_baselines_type', 'quality_baselines', ['baseline_type'])

# Create quality_regressions table
op.create_table(
    'quality_regressions',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('baseline_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('workflow_execution_id', postgresql.UUID(as_uuid=True), nullable=True),
    sa.Column('regression_type', regression_type_enum, nullable=False),
    sa.Column('severity', regression_severity_enum, nullable=False),
    sa.Column('detected_at', sa.DateTime(), nullable=False),
    sa.Column('metric_name', sa.String(100), nullable=False),
    sa.Column('baseline_value', sa.Float(), nullable=False),
    sa.Column('current_value', sa.Float(), nullable=False),
    sa.Column('delta_percent', sa.Float(), nullable=False),
    sa.Column('sample_size', sa.Integer(), nullable=False),
    sa.Column('action_taken', regression_action_enum, nullable=False),
    sa.Column('resolved_at', sa.DateTime(), nullable=True),
    sa.Column('resolution_notes', sa.Text(), nullable=True),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.Column('updated_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['baseline_id'], ['quality_baselines.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['workflow_execution_id'], ['workflow_executions.id'], ondelete='SET NULL'),
)

# Create indexes for quality_regressions
op.create_index('ix_quality_regressions_id', 'quality_regressions', ['id'])
op.create_index('ix_quality_regressions_agent_id', 'quality_regressions', ['agent_id'])
op.create_index('ix_quality_regressions_customer_id', 'quality_regressions', ['customer_id'])
op.create_index('ix_quality_regressions_baseline_id', 'quality_regressions', ['baseline_id'])
op.create_index('ix_quality_regressions_workflow_execution_id', 'quality_regressions', ['workflow_execution_id'])
```

```
    op.create_index('ix_quality_regressions_regression_type', 'quality_regressions', ['regression_type'])
    op.create_index('ix_quality_regressions_severity', 'quality_regressions', ['severity'])
    op.create_index('ix_quality_regressions_action_taken', 'quality_regressions', ['action_taken'])
    op.create_index('ix_quality_regressions_detected_at', 'quality_regressions', ['detected_at'])
    op.create_index('ix_quality_regressions_agent_severity', 'quality_regressions', ['agent_id', 'severity'])
    op.create_index('ix_quality_regressions_customer_type', 'quality_regressions', ['customer_id', 'regression_type'])
    op.create_index('ix_quality_regressions_baseline', 'quality_regressions', ['baseline_id'])
    op.create_index('ix_quality_regressions_workflow', 'quality_regressions', ['workflow_execution_id'])
    op.create_index('ix_quality_regressions_unresolved', 'quality_regressions', ['resolved_at'])


def downgrade() -> None:
    """Drop application schema tables"""

    # Drop quality_regressions table and indexes
    op.drop_index('ix_quality_regressions_unresolved', 'quality_regressions')
    op.drop_index('ix_quality_regressions_workflow', 'quality_regressions')
    op.drop_index('ix_quality_regressions_baseline', 'quality_regressions')
    op.drop_index('ix_quality_regressions_customer_type', 'quality_regressions')
    op.drop_index('ix_quality_regressions_agent_severity', 'quality_regressions')
    op.drop_index('ix_quality_regressions_detected_at', 'quality_regressions')
    op.drop_index('ix_quality_regressions_action_taken', 'quality_regressions')
    op.drop_index('ix_quality_regressions_severity', 'quality_regressions')
    op.drop_index('ix_quality_regressions_regression_type', 'quality_regressions')
    op.drop_index('ix_quality_regressions_workflow_execution_id', 'quality_regressions')
    op.drop_index('ix_quality_regressions_baseline_id', 'quality_regressions')
    op.drop_index('ix_quality_regressions_customer_id', 'quality_regressions')
    op.drop_index('ix_quality_regressions_agent_id', 'quality_regressions')
    op.drop_index('ix_quality_regressions_id', 'quality_regressions')
    op.drop_table('quality_regressions')

    # Drop quality_baselines table and indexes
    op.drop_index('ix_quality_baselines_type', 'quality_baselines')
    op.drop_index('ix_quality_baselines_customer_model', 'quality_baselines')
    op.drop_index('ix_quality_baselines_agent_model', 'quality_baselines')
    op.drop_index('ix_quality_baselines_established_at', 'quality_baselines')
    op.drop_index('ix_quality_baselines_baseline_type', 'quality_baselines')
    op.drop_index('ix_quality_baselines_model_name', 'quality_baselines')
    op.drop_index('ix_quality_baselines_customer_id', 'quality_baselines')
    op.drop_index('ix_quality_baselines_agent_id', 'quality_baselines')
    op.drop_index('ix_quality_baselines_id', 'quality_baselines')
    op.drop_table('quality_baselines')

    # Drop quality_metrics table and indexes
```

```
    op.drop_index('ix_quality_metrics_model_timestamp', 'quality_metrics')
    op.drop_index('ix_quality_metrics_customer_model', 'quality_metrics')
    op.drop_index('ix_quality_metrics_agent_model', 'quality_metrics')
    op.drop_index('ix_quality_metrics_timestamp', 'quality_metrics')
    op.drop_index('ix_quality_metrics_overall_quality_score', 'quality_metrics')
    op.drop_index('ix_quality_metrics_hallucination_detected', 'quality_metrics')
    op.drop_index('ix_quality_metrics_model_name', 'quality_metrics')
    op.drop_index('ix_quality_metrics_request_id', 'quality_metrics')
    op.drop_index('ix_quality_metrics_customer_id', 'quality_metrics')
    op.drop_index('ix_quality_metrics_agent_id', 'quality_metrics')
    op.drop_index('ix_quality_metrics_id', 'quality_metrics')
    op.drop_table('quality_metrics')

    # Drop ENUM types
    sa.Enum(name='regression_action').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='regression_severity').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='regression_type').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='baseline_type').drop(op.get_bind(), checkfirst=True)
```

🚀 EXECUTION STEPS

Step 1: Create Models File

```
bash

cd ~/optiinfra/shared/database/models

# Create application_schema.py
cat > application_schema.py << 'EOF'
[Copy the entire application_schema.py content from FILE 1 above]
EOF

# Verify file created
ls -lh application_schema.py
# Expected: ~550 lines
```

Step 2: Update Core Models

```
bash
```

```
cd ~/optiinfra/shared/database/models

# Open core.py and add relationships
nano core.py

# Add to Agent class:
#   quality_metrics = relationship(...)
#   quality_baselines = relationship(...)
#   quality_regressions = relationship(...)
#
# Add to Customer class:
#   quality_metrics = relationship(...)
#   quality_baselines = relationship(...)
#   quality_regressions = relationship(...)
```

Step 3: Update Model Imports

```
bash

cd ~/optiinfra/shared/database/models

# Update __init__.py with application models
cat > __init__.py << 'EOF'
[Copy the entire __init__.py content from FILE 3 above]
EOF
```

Step 4: Create Migration File

```
bash

cd ~/optiinfra/shared/database/migrations/versions

# Create migration file
cat > 005_application_schema.py << 'EOF'
[Copy the entire migration content from FILE 4 above]
EOF

# Verify file created
ls -lh 005_application_schema.py
# Expected: ~220 lines
```

Step 5: Test Imports

```bash
cd ~/optiinfra

# Test if new models can be imported
python << 'EOF'
from shared.database.models.application_schema import (
    QualityMetric,
    QualityBaseline,
    QualityRegression,
    BaselineType,
    RegressionType,
    RegressionSeverity,
    RegressionAction
)
print('✅ All application models import successfully')

# Test enums
print(f'✅ BaselineType has {len(BaselineType)} values')
print(f'✅ RegressionType has {len(RegressionType)} values')
print(f'✅ RegressionSeverity has {len(RegressionSeverity)} values')
print(f'✅ RegressionAction has {len(RegressionAction)} values')
EOF

# Expected output:
# ✅ All application models import successfully
# ✅ BaselineType has 3 values
# ✅ RegressionType has 4 values
# ✅ RegressionSeverity has 4 values
# ✅ RegressionAction has 4 values
```

Step 6: Run Alembic Migration

```bash
```

```
cd ~/optiinfra/shared/database

# Check current migration state
alembic current
# Expected: 004_resource_schema (current)

# Run the new migration
alembic upgrade head

# Expected output:
# INFO  [alembic.runtime.migration] Running upgrade 004_resource_schema -> 005_application_schema, Create applicati
```

Step 7: Verify Tables Created

bash

```
# Connect to PostgreSQL
psql postgresql://optiinfra:password@localhost:5432/optiinfra

# List all tables (should now be 18 tables)
\dt

# Expected output (15 from previous + 3 new):
#  quality_metrics
#  quality_baselines
#  quality_regressions
#  resource_metrics
#  scaling_events
#  workflow_executions
#  workflow_steps
#  workflow_artifacts
#  agent_configs
#  agent_states
#  agent_capabilities
#  agent_metrics
#  customers
#  agents
#  events
#  recommendations
#  approvals
#  optimizations

# Describe quality_metrics table
\d quality_metrics

# Expected: Should show all columns, indexes, foreign keys

# Describe quality_baselines table
\d quality_baselines

# Expected: Should show all columns, relationships

# Describe quality_regressions table
\d quality_regressions

# Expected: Should show all columns, foreign keys to baselines and workflows

# Check new enums were created
\dT+
```

```
# Expected: Should see baseline_type, regression_type, regression_severity, regression_action

# Exit
\q
```

Step 8: Verify Relationships Work

bash

```
cd ~/optiinfra

# Test relationships in Python
python << 'EOF'
from shared.database.models import Agent, Customer, QualityMetric, QualityBaseline, QualityRegression
from shared.database.session import SessionLocal

db = SessionLocal()

# Get a test agent
agent = db.query(Agent).first()
print(f"✅ Agent: {agent.name}")
print(f"   - Has quality_metrics relationship: {hasattr(agent, 'quality_metrics')}")
print(f"   - Has quality_baselines relationship: {hasattr(agent, 'quality_baselines')}")
print(f"   - Has quality_regressions relationship: {hasattr(agent, 'quality_regressions')}")

# Get a test customer
customer = db.query(Customer).first()
print(f"✅ Customer: {customer.name}")
print(f"   - Has quality_metrics relationship: {hasattr(customer, 'quality_metrics')}")
print(f"   - Has quality_baselines relationship: {hasattr(customer, 'quality_baselines')}")
print(f"   - Has quality_regressions relationship: {hasattr(customer, 'quality_regressions')}")

# Check models
print(f"✅ QualityMetric model loaded")
print(f"   - Has agent relationship: {hasattr(QualityMetric, 'agent')}")
print(f"   - Has customer relationship: {hasattr(QualityMetric, 'customer')}")

print(f"✅ QualityBaseline model loaded")
print(f"   - Has agent relationship: {hasattr(QualityBaseline, 'agent')}")
print(f"   - Has customer relationship: {hasattr(QualityBaseline, 'customer')}")
print(f"   - Has regressions relationship: {hasattr(QualityBaseline, 'regressions')}")

print(f"✅ QualityRegression model loaded")
print(f"   - Has agent relationship: {hasattr(QualityRegression, 'agent')}")
print(f"   - Has baseline relationship: {hasattr(QualityRegression, 'baseline')}")
print(f"   - Has workflow relationship: {hasattr(QualityRegression, 'workflow')}")

db.close()
print("\n✅ All relationship verifications passed!")
EOF
```

```
# Expected output similar to previous verifications
```

🎯 PART 1 SUMMARY

Files Created/Modified:

✅ **shared/database/models/application_schema.py** - 3 new models + 4 enums (~550 lines)
✅ **shared/database/models/core.py** - Modified (added 6 relationships)
✅ **shared/database/models/init.py** - Modified (exports)
✅ **shared/database/migrations/versions/005_application_schema.py** - Migration (~220 lines)

What Works Now:

✅ Can import all new models in Python
✅ Can run Alembic migration
✅ 3 new tables created in PostgreSQL
✅ All foreign keys working (cascades configured)
✅ All indexes created (35+ indexes)
✅ Agent-quality relationship working
✅ Customer-quality relationship working
✅ Regression-baseline-workflow relationships working

Database Status:

```
Total Tables: 18 (100% PostgreSQL schema COMPLETE!) 🎉
├── Core (0.2a): 6 tables ✅
├── Agent State (0.2b): 4 tables ✅
├── Workflow History (0.2c): 3 tables ✅
├── Resource Schema (0.2d): 2 tables ✅
└── Application Schema (0.2e): 3 tables ✅

Total Enums: 21
Total Indexes: 115+
Total Relationships: 35+
```

🎊 MILESTONE ACHIEVED:

PostgreSQL schema is 100% COMPLETE!

All 4 agents now have complete database support:

- ✅ Cost Agent (core tables + metrics)
- ✅ Performance Agent (core tables + metrics)
- ✅ Resource Agent (resource_metrics, scaling_events)
- ✅ Application Agent (quality_metrics, baselines, regressions)

What's in Part 2:

- Seed data (quality metrics, baselines, regressions)
- Test fixtures (sample quality data, regression scenarios)
- Test cases (14-16 comprehensive tests)
- Integration tests (complete regression detection flow)
- All validation commands
- Troubleshooting guide
- Success criteria checklist
- Git commit instructions
- **Celebration of PostgreSQL completion!** 🎉

⏭️ NEXT STEP

Download PART 2 to get:

- Complete seed data implementation (realistic quality metrics)
- Full test suite (14-16 tests covering all scenarios)
- Step-by-step validation
- Troubleshooting for common issues
- Final checklist
- **PostgreSQL 100% completion celebration!**

File name: `FOUNDATION-0.2e-Application-Schema-PART2-Testing.md`

📥 HOW TO DOWNLOAD THIS FILE

1. Click the **"Copy"** dropdown button at the top of this artifact

2. Select **"Download as txt"**

3. Save as: `FOUNDATION-0.2e-Application-Schema-PART1-Code.md`

✅ VERIFICATION CHECKLIST (Part 1)

Before moving to Part 2, verify:

- [ ] `application_schema.py` created with 3 models
- [ ] `core.py` updated with 6 new relationships
- [ ] `__init__.py` exports all application models
- [ ] Migration file `005_application_schema.py` created
- [ ] Migration runs successfully (`alembic upgrade head`)
- [ ] 3 new tables exist in PostgreSQL
- [ ] 4 new enum types created
- [ ] All indexes created
- [ ] Python imports work (no errors)
- [ ] Relationships work (verified in Python)

If all checked, you're ready for PART 2! ✅

✅ **PART 1 COMPLETE! Ready for PART 2 and PostgreSQL completion?** 🚀 🎉 lete='CASCADE'),)

```
# Create indexes for quality_metrics
op.create_index('ix_quality_metrics_id', 'quality_metrics', ['id'])
op.create_index('ix_quality_metrics_agent_id', 'quality_metrics', ['agent_id'])
op.create_index('ix_quality_metrics_customer_id', 'quality_metrics', ['customer_id'])
op.create_index('ix_quality_metrics_request_id', 'quality_metrics', ['request_id'], unique=True)
op.create_index('ix_quality_metrics_model_name', 'quality_metrics', ['model_name'])
op.create_index('ix_quality_metrics_hallucination_detected', 'quality_metrics', ['hallucination_detected'])
op.create_index('ix_quality_metrics_overall_quality_score', 'quality_metrics', ['overall_quality_score'])
op.create_index('ix_quality_metrics_timestamp', 'quality_metrics', ['timestamp'])
op.create_index('ix_quality_metrics_agent_model', 'quality_metrics', ['agent_id', 'model_name'])
op.create_index('ix_quality_metrics_customer_model', 'quality_metrics', ['customer_id', 'model_name'])
op.create_index('ix_quality_metrics_model_timestamp', 'quality_metrics', ['model_name', 'timestamp'])

# Create quality_baselines table
op.create_table(
    'quality_baselines',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('model_name', sa.String(100), nullable=False),
    sa.Column('model_version', sa.String(50), nullable=False),
    sa.Column('baseline_type', baseline_type_enum, nullable=False),
    sa.Column('sample_size', sa.Integer(), nullable=False),
    sa.Column('avg_relevance_score', sa.Float(), nullable=True),
    sa.Column('avg_coherence_score', sa.Float(), nullable=True),
    sa.Column('avg_factuality_score', sa.Float(), nullable=True),
    sa.Column('avg_overall_score', sa.Float(), nullable=False),
    sa.Column('p95_latency_ms', sa.Float(), nullable=True),
    sa.Column('established_at', sa.DateTime(), nullable=False),
    sa.Column('valid_until', sa.DateTime(), nullable=True),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], onde
```